Solist: A Lightweight Multi-Overlay Structure for Wireless Sensor Networks

Yann Busnel, Marin Bertier, Anne-Marie Kermarrec

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

SOLIST: *A Lightweight Multi-Overlay Structure for Wireless Sensor Networks*

Yann Busnel — Marin Bertier — Anne-Marie Kermarrec

N° 6404

Décembre 2007

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--6404--FR+ENG

SOLIST: A Lightweight Multi-Overlay Structure for Wireless Sensor Networks

Yann Busnel*, Marin Bertier†, Anne-Marie Kermarrec‡

Thème NUM — Systèmes numériques
Projets Asap

Rapport de recherche n° 6404 — Décembre 2007 — 27 pages

Abstract: Sensor networks have recently received an increasing interest both in industry and academia and are applicable to a wide range of distributed applications. In such networks, sensors organize themselves to collect or disseminate information in the network, routing messages to specific sensors, *etc.* Given the limited resources of those devices, energy consumption is a first class concern.

At the core of data management in distributed systems, lie some basic functionalities such as broadcast or anycast. In this paper, we focus on providing a suite of *-cast (anycast, k-cast, broadcast) functionalities in a fully decentralized manner. More specifically, we present the design and evaluation of SOLIST, implementing the *-cast suite for wireless sensor networks. SOLIST is fully decentralized, and let the sensors get organized into a multi-layer structure, largely inspired from structured peer-to-peer systems, yet limiting the overall energy consumption. A type is associated to each sensor, and the *-cast functionalities are implemented at a type granularity regardless of the number of types and their distribution within the network. This enables to reach any (anycast), k (k-cast), or all (broadcast) sensors of a given type in a large-scale network. Sensor nodes of the same type are logically clustered and each of those types remains reachable from any point in the system as long as the network is connected. We evaluate SOLIST through simulation and show that SOLIST provides those functionalities while achieving a reasonable trade-off between performance and energy consumption.

Key-words: Wireless sensor network, structured network, diffusion primitives, anycast, multicast, peer-to-peer, energy-based

* IRISA / Université de Rennes 1
† IRISA / INSA Rennes
‡ IRISA / INRIA Rennes – Bretagne Atlantique

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex (France)
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

SOLIST : Structure multicouche pair-à-pair à faible consommation pour les réseaux de capteurs sans-fil.

Résumé : Les réseaux de capteurs sans-fil (RCsF) connaissent depuis quelques années un intérêt croissant, tant dans la recherche que dans l'industrie. Ceux-ci ouvrent une nouvelle voie aux applications réparties dans des milieux jusqu'alors impraticables, les capteurs s'auto-organisant eux-mêmes afin de fournir des fonctionnalités telles que la collecte ou la dissémination d'information sur et dans le réseau, le routage de messages à des capteurs spécifiques, etc. En raison de leur petite taille, les ressources disponibles sur un capteur sont très limitées, et la gestion d'énergie devient donc une préoccupation de premier ordre.

Le cœur des systèmes de gestion de données dans les systèmes répartis repose sur certaines fonctionnalités de bases comme la diffusion ou la recherche de capteurs particuliers. Dans cet article, nous proposons la mise en œuvre décentralisée d'une collection de primitive appelées *-cast (anycast, k-cast et broadcast). Pour cela, nous présentons et évaluons SOLIST, une structure multicouche, largement inspiré des réseaux pair-à-pair structurés, limitant la consommation d'énergie dans le cadre des RCsF. Un type est associé à chaque capteur, et les primitives de *-cast sont mises en œuvre à la granularité des types, sans pour autant nécessiter une connaissance de ces types à priori, ni de leur répartition dans le réseau. Nous avons évalué SOLIST par simulation et montré que cette structure fournit les primitives de *-cast en atteignant un bon compromis entre performance et consommation d'énergie.

Mots-clés : Réseaux de capteurs, réseaux structurés, primitives de diffusion, pair-à-pair, économie d'énergie

1 Introduction

Looking for a specific information in a large-scale network with no underlying structure is as difficult as searching a needle in a haystack without any metal detector. This is somehow illustrated by the inefficiency of unstructured peer-to-peer networks to search rare items. While this is not such an issue in wired networks, flooding the network is crippling in wireless sensor networks (WSN) where energy saving dominates. Effectively, WSNs consist of a high number of small entities (usually call *MEMS*: *Micro-Electro Mechanical Systems*), having much less capabilities than common computers. Due to their tiny size, this MEMS, equipped of a wireless communication appliance and called *sensors* in the following, possess slim resources in terms of memory, CPU, *etc.* [2, 17]. Energy consumption more specifically is a first class concern in WSN deployment.

As opposed to general-purpose large-scale distributed systems, WSN are deployed and configured usually to fulfil a specific application needs. Example of such applications are monitoring, inventory, data aggregation, *etc.* Yet some basic functionalities are common to a large number of those applications. At the heart of data management, we have identified a set of communication primitives that can be seen as basic building blocks for those applications. We call the **-cast suite*, this set of functionalities. This suite consists in broadcast, anycast and k-cast primitives. We consider applications manipulating some subsets of sensors. To each subset is associated a specific type. Then, each sensor is belonged to a type, which can be dynamically changed with respect to application needs or execution. An *Anycast* query aims at reaching a specific entity of a given type among sensors. A *broadcast* aims at reaching all entities of a given type in a network. A *k-cast* operation aims at reaching k entities of a given type[1]. To illustrate our purpose, consider an inventory application, where each sensor represents a physical item of a given type. Sending a message to all the sensors of a given type or querying the system to know whether there still exist an instance of a given type, are standard operations. The aforementioned *-cast suite would provide the means to implement easily such operations.

In this paper, we present the design and evaluation of SOLIST (*Self-Organized Large-scale and lightweight Information-based Sensor Technology*), a structured overlay network for WSN providing an efficient *-cast suite. Based on a simple common interface (group clustering and *-cast), SOLIST provides a generic infrastructure while ensuring low energy consumption. In order to implement the *-cast operations, SOLIST relies on a lightweight multi-layer structure. Sensors are clustered according to their type into specific layers. k-cast and broadcast capabilities are implemented at each layer. In addition, SOLIST offers an efficient anycast mechanism. This functionality can be used in standalone to implement the anycast operation as well as to get an entry point to a specific type.

The rest of the paper is organized as follows. In Section 2 and 3, we introduce respectively the system model and the generic interface. The SOLIST design is presented in Section 4 and the *-cast suite implementation in Section 5. In order to evaluate SOLIST, we conducted simulations in worst case scenarios and compared it with traditional approaches. We present the experimentation results in Section 6. Finally, Section 7 briefly surveys the related works before concluding in Section 8.

[1] where k is a given parameter of this primitive.

2 System model

We consider a network composed of n wireless sensors, distributed in a given geographical area. Although we do not make specific assumptions on the distribution of the sensors, we assume a connected network. We also assume that sensors are not mobile. They may however become unavailable, due to a failure or the fact that they have run out of battery. Nodes can communicate only by 1-hop broadcast with nodes in their transmission range, through an ideal communication layer: no collision and no message loss are considered. Furthermore, the propagation delay between two neighbours has to be bound, and this last value is known.

Each node is aware of its relative geographical position in a virtual coordinate space as well as the size and the network borders coordinates. We do not provide in this paper a method to construct such a coordinate system. Instead we consider a static network and nodes' coordinates can be computed locally when they *join* the network as in [10, 13]. Each node is assigned a given type and knows its local type. No predefined set of types is required.

In order to discover its own neighbourhood, each node broadcasts locally and periodically `Hello` messages, called *beacon* in the following. Then, to implement a multi-hop communication, any geographic routing protocol can be used, as a node may send messages to another one by knowing only its relative position. In our implementation, we use the GPSR [9] routing protocol which allows to reach efficiency the nearest node of a point identified by virtual coordinates.

3 The *-cast suite

A large majority of distributed applications rely on some basic functionalities. More specifically in WSN, nodes may need to be accessed depending on their category. SOLIST provides a basic set of functionalities, identified as *-cast and relies to this end on a group-based structure. Each node is assigned a given *type* representing its state at a specific time. Note that this may target static type (if sensors are heterogeneous for example) as well as dynamic (related to the level of remaining battery, or sensed data). A node may also be without type and therefore only contribute to the global connectivity and communication. SOLIST ensures that every node of a same type, are *clustered* together dynamically.

We identified as core functionalities the three following operations:

Anycast ANYCAST(*type*);

This service aims at reaching one node, if any, of a specific type.

For example, this functionality may be used to discover if one instance of a given exists in a system and localize this instance.

k-cast KCAST(*type*, k);

This service aims at reaching k nodes, if they exist, which belong to a specific type. If this group contains more than k nodes, this function return TRUE*, and* FALSE *otherwise. This operation provides a direct access to those k instances although we could consider attaching more information to the reply message.*

For example, stock managers usually need to know if objects of a specific type exist in sufficient quantity. Another example is a fire monitoring

application; users can need to be aware if 10 sensors have sensed a temperature greater than 40 degrees, or a hygrometry value lower than 2 %vol. Also, firemen may need the average value of temperature and hygrometry for instance, on a random sample of common type nodes. k-cast might be useful in such examples.

Broadcast BROADCAST(*type*);
This service aims at reaching all nodes belonging to a specific type. This function may return the number of nodes of the group or other information on these nodes.
As SOLIST can broadcast to only a specific type of nodes, this service can be view as a *multicast* one. Such a functionality may be used either to disseminate a message to all nodes of a given type as in a publish-subscribe system for example. It can also be used to enumerate the number of nodes of a given type.

Each node also implements the following join and leave operations.

Join JOIN(*type*); When a node joins an existing WSN, we assume that it can contact a node already belonging to the network. It commonly executes several tasks to initialize itself, inform its direct neighbourhood of its presence, join the various structures it has to belongs to, *etc.* We will provide more details later in the paper.

Leave LEAVE(); When a node leaves the network voluntarily, it usually executes a leave procedure to avoid structure inconsistency, update information on its neighbours, *etc.* Whenever a node leaves the network without notice, this is considered as a failure.

Whenever a node changes its type, it performs first a leave operation to quit the former group and then a join operation to become part of the group associated to the new type, if there is.

As we mentioned previously, *-cast, and then SOLIST, may be used in the context of many WSN applications. For instance, we can cite several classical applications in which SOLIST would be relevant:

Monitoring In such applications, users may need to know how many supervised entities are in a specific state at a specific time. By using dynamic grouping (to cluster sensors in the same state) and *-cast in a specific group, we can obtain most of information without using a particular query language as in [11];

Diffusion This application is inherent of the *-cast suite. k-cast and broadcast provide respectively a bound and unbound multicast functionalities. To reach several groups of nodes in the network, users send a broadcast query in each of these groups;

Aggregation By using an inverted k-cast/broadcast operation, information on all nodes in a group can be collected and aggregated in the reply to the initiator;

Inventory management In such applications, stock managers have to be able to answer of three essential queries: are there any instances of a specific

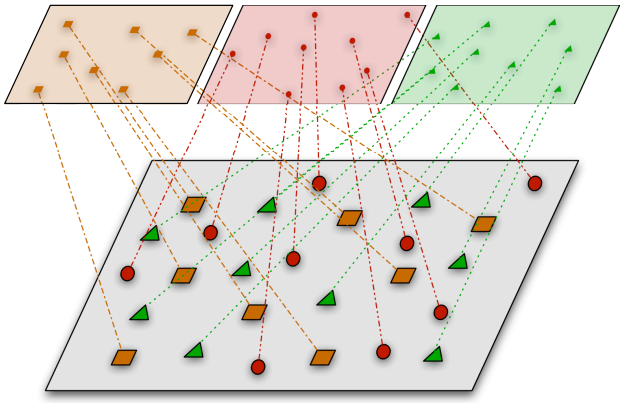

Figure 1: Projection into layers

kind of items? Is the quantity sufficient? How many instances of such item exist? The *-cast suite directly answer those questions by respectively using anycast, k-cast and broadcast;

Publish/subscribe By using dynamic type, joining a group can be seen as a subscription to a topic, and sending a broadcast correspond to a publication [7, 19]. In the context of WSN, this can be used to implement alerts for instance.

4 SOLIST core

In this section, we present the SOLIST architecture. The *-cast services in SOLIST are presented in the following section. These services are based on the SOLIST structure in order to combine efficiency and energy saving.

4.1 A multi-layer structure

In order to provide the *-cast suite, SOLIST relies on a multi-layer structure. All nodes belong to the basic layer, implemented to ensure the global connectivity and the geographical routing. We use the GPSR [9] protocol to route in this layer. This layer is denoted *routing layer* in the following. It enables to reach any destination based on its virtual coordinates[2].

On top of the basic layer, SOLIST implements a lightweight specific overlay per group. We specify the building and functioning of this overlay in Section 4.2.

The multi-layer structure provides a logical clustering of the network. For example, in Figure 1, n nodes are spread between three different types. For each type corresponds a specific overlay (*i.e.* the triangle overlay, the circle and the square ones) on the top of the routing layer. Communications between two nodes of a specific overlay are implemented using the routing layer.

[2]Those virtual coordinates is provided as introduced in Section 2

4.2 Layer structure: LIGH-*t*-LAYER

SOLIST provides a lightweight multi-layer structure obtained by dividing each subspace into several rectangle-shapes. As we obseserved that WSNs have a lot of similitude with peer-to-peer (P2P) systems [3] (in term of properties and functionalities), we propose a structure inspired from a standard P2P Distributed Hash Table (DHT) namely CAN [15]. In [4], authors presents an evaluation of P2P structured overlay for multicast to implement each mini-overlay. CAN provides an ideal structure for the targeted functionalities in a WSN. CAN splits the whole space into logical responsibility areas, evenly spread between nodes in the system. Each node in the system has a virtual neighbourhood, corresponding to the adjacent areas of its own zone. We slightly modify this structure to match the WSN context. The main difference is that communication and energy constraints, specific to sensors, must be considered. Therefore, we keep only the virtual space partition, given that the localization of a node is the same between this layer and the routing one. Thus, each node in the WSN has the same virtual coordinate for every overlay and the routing layer. Responsibility areas are only used to implement efficient k-cast and broadcast in group. We denote this structure by LIGH-*t*-LAYER in the following[3].

The network is built gradually. The first joining node of a given type is *responsible* for the whole network space for this specific type LIGH-*t*-LAYER. When another node arrives in the network, it contacts the node responsible of the area where it belongs, using a greedy minimization of the distance into the corresponding LIGH-*t*-LAYER. The area is then split so that the area responsability is shared between the two nodes as it is done in CAN. A new frontier is created between the two areas. This frontier is stamped by an identifier in order to keep the area creation order. This last information is used in case of node leave or fail, to reorganize the structure as explain below. In short, at each arrival, the node responsible of the area splits its own area, and creates a new stamped frontier between it and the arrival node.

Each node, belonging to a specific layer, has to maintain a list of LIGH-*t*-LAYER neighbours, called $view_t$. Each $view_t$ entry has to contain (1) the neighbour's id; (2) the neighbour's coordinates; (3) the begin and end coordinates of the frontier; (4) the frontier stamp.

Figure 2 presents the different cases of evolution of a LIGH-*t*-LAYER structure from the start. Node A joins first and is responsible of the whole space. In Figure 2a, B contacts A, which in turn splits its area in two and sends to B the coordinates of the B's area and of the new frontier common to both. Then, at C's arrival, B splits its own area and stamps the new frontier by a strictly greater identifier as in Figure 2b. Figure 2c presents the state of the layer structure with 9 nodes, from A to I. Here, the biggest stamp is 5.

The bottom part of Figure 2 presents how to keep a consistent structure in case of a departure (Figure 2d) or a failure (Figure 2e). Departures and failures are handled the same way. When a node leaves a layer (in case of network departure, or type changing in SOLIST context), it sends its $view_t$ to the nodes across the last created frontier. These nodes become responsible of the union of the two areas, update their $view_t$ with the potential new neighbours, extend frontiers and finally send the updated information to the corresponding neighbours.

[3] t corresponds to the specific type t of nodes clustered in this light layer.

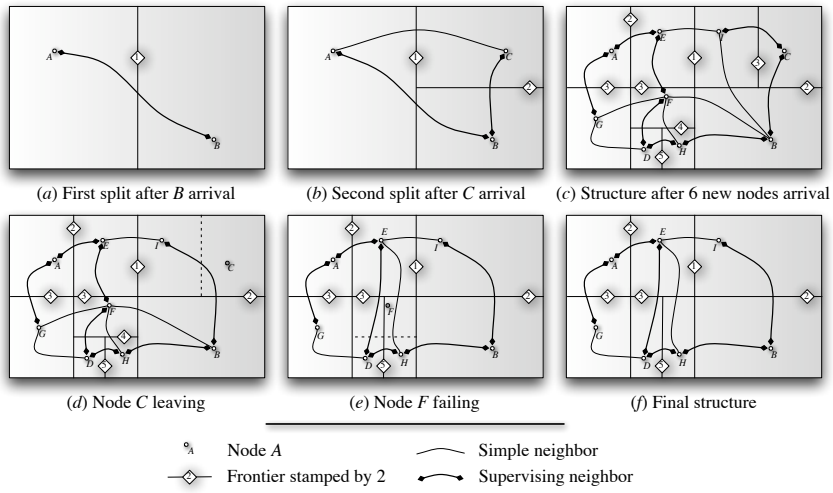

Figure 2: Evolution of a LIGH-t-LAYER structure following different kinds of event

For instance, node C, in Figure 2d, has to leave the layer. C sends its $view_t$ to node I across frontier 3 (the highest frontier stamp). I becomes the responsible of the merged area: C's and I's one. I updates its $view_t$ with an extended frontier in common with B. Finally, I sends this information to node B.

Failure detection is provided by a survey mechanism, as introduced in Section 2. At each joining step, the two nodes involved in the splitting initiate a symmetric supervising. Periodically, every node in the layer sends an *iAmAlive* message to its supervisor. If a supervisor does not receive the *iAmAlive* message from one of its supervised nodes, it checks upon the node by sending an *areYouAlive* message to this last. If no answer is received after a predefined timeout, the node is considered faulty, and removed of the system.

In case of failure, as in Figure 2e, node F is removed from SOLIST. F can't send its own $view_t$ across the last creating frontier, as in leaving procedure. Although, node D, which is the F's supervisor, sends a *collectView* request to be aware of F's $view_t$. This request is routed around the F's area and comes back to D. With this information, D just has to execute the leaving procedure, by contacting all nodes across the same frontier (here, node H) and merge the corresponding F's sub-area with its own one. Figure 2f presents the LIGH-t-LAYER structure state after those actions are performed.

4.3 Linking up the world: Entry points

A challenging issue is how to reach a specific LIGH-t-LAYER from any node, without any information about it except the type t identifier. We map a logical namespace on the top of the routing layer, as proposed in recent works [14]. We use common hash functions so that all nodes share a common coordinate set in this namespace. The aim of these hash functions is to spread uniformly

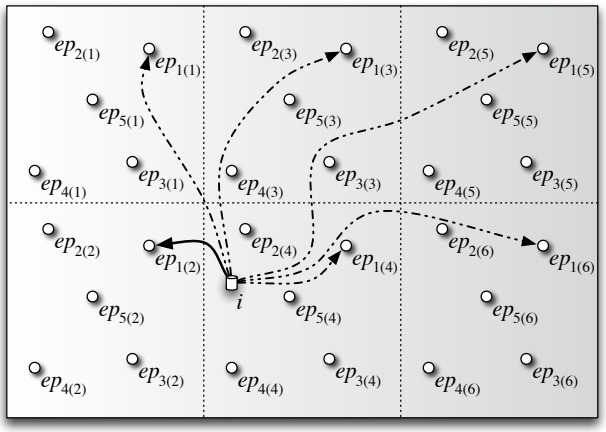

Figure 3: Example of a 3×2 cell namespace divisions.
Node i sends its request for layer 1 to the nearest entry point: $ep_{1(2)}$

coordinates into space according to the number of types. In the following, we call these coordinates as *entry points*. Let t be the request type identifier of the layer searched for. Let $f_x : \mathbb{N} \mapsto [0;1[$ and $f_y : \mathbb{N} \mapsto [0;1[$ two hash functions and wsX, wsY the size of the virtual coordinate space respectively horizontally and vertically. The entry point coordinates (denoted by ep_t) corresponding to this layer are computed as the following Equation 1:

$$ep_t = (x, y) \text{ where } \begin{cases} x &= f_x(t) \times wsX \\ y &= f_y(t) \times wsY \end{cases} \quad (1)$$

As f_x and f_y are common for all nodes, the entry point coordinates are unique for a specific type in the network. An entry point is represented by its hashed coordinates; however there might not be any node at this specific location in the whole space. Therefore, we shall make no distinction between a point in the virtual space and the actual nearest node, called *entry point* as well in the following. This nearest node of ep_t has to know at least one node of type t (and so, to be aware of the existence of this LIGH-t-LAYER). If this LIGH-t-LAYER exists, it will be able to reply with the identifier and coordinates of a node belonging to this layer. In the following, this last node (known by an entry point) is called *contact node*.

In order to balance the load of the nearest node to an entry point and to avoid that some requests have to traverse the whole network to reach an entry point, the namespace is split according to a grid: m horizontal divisions and n vertical divisions, according to x and y coordinates. The namespace is mapped to each division, which are called *cells* in the following. When a node has to reach a specific layer, it sends its request to the nearest corresponding entry point, which should reply with some information about a contact node in the specific layer.

Figure 3 presents a $m \times n = 3 \times 2$ topology with 5 identified types. For each cell, the 5 entry points (et_1 to et_5) have the same relative position. When

a node is willing to access a specific layer, it sends its request to the nearest entry point of the given type. Here, node i sends a request to access the layer associated to type 1. i is aware of all the 6 entry points (dash and solid arrow) and sends its request to the nearest one (here, $ep_{1(2)}$ represented by a solid arrow). Let $d(\cdot,\cdot) : (\mathbb{R}\times\mathbb{R})^2 \mapsto \mathbb{R}$ the distance between two points in the virtual space and csX, csY the size of the cells respectively horizontally and vertically. To find the nearest entry point, node i has to do some computations, presented in Equation 2. Let the two sets:

$$\begin{cases} \Gamma_{x,t} = \{(k_x + f_x(t)) \times csX | k_x \in [0..m-1]\} \\ \Gamma_{y,t} = \{(k_y + f_y(t)) \times csY | k_y \in [0..n-1]\} \end{cases}$$

$$\begin{aligned} ep_t &= (x,y) \\ \text{ST } d(i,(x,y)) &= \min_{\substack{x' \in \Gamma_{x,t} \\ y' \in \Gamma_{y,t}}} d(i,(x',y')) \end{aligned} \tag{2}$$

In order to update the knowledge of entry points for this searching layer mechanism, when a node joins a specific LIGH-t-LAYER, it sends its arrival information to the nearest entry point corresponding to the type t. So, this node is aware of the nearest node belonging to the LIGH-t-LAYER layer, and can send information about a close node when a search layer request arrives. Likewise, in case of failure or departure, entry points are informed in order to update their contact nodes links. As entry points choose locally their nearest contact node, the leaving/faulty node is not aware on which entry point it corresponds to. So, all entry points have to be informed of such changes in the network.

4.4 Requirement summary

In this section, we summarize the state that each node needs to maintain:

- a unique identifier (id);
- its coordinates in the virtual space (x_{id}, y_{id});
- the size of this virtual space (wsX, wsY);
- its type if needed (t_{id});
- two common hash functions (f_x, f_y) and a distance function d, in order to compute the positions of entry points;
- the number of cells (m,n)[4];

If a node has a type t defined, it has to keep a $view_t$ up-to-date (*cf.* Section 4.2 for details).

If a node is an entry point of type t', it has to keep the identifier and coordinates in the virtual space of one node in the corresponding LIGH-t'-LAYER.

[4]The cell size csX and csY is computed dynamically as wsX/m and wsY/n

5 *-cast in SOLIST

We now present how to implement the *-cast functionalities efficiently with respect to energy consumption and reliability[5].

5.1 Anycasting a group

A node sends an anycast request in order to reach *any* node of a specific type. We observe that the searching mechanism within a layer, presented in Section 4.3, can actually be applied directly to implement an anycast request. This searching mechanism returns to the request transmitter the identifier and the coordinates of a contact node, which belong to the specific LIGH-t-LAYER, or `NOT_FOUND` if this layer does not exist. We call this layer searching mechanism *anycast* in the following.

As the mechanism presented in Section 4.3 updates continuously the information located on the entry points, an anycast reply corresponds to the nearest contact node of the entry point. Although, the smaller cells size in SOLIST, the nearer contact node reply.

Considering the reliability of this anycast mechanism, out of date information can only appear in two cases. When the first node of a type joins the network, it has to inform all the entry points in the network that a new LIGH-t-LAYER is created. Also, during the period between the creation and the reception of information on all entry points, a not yet informed entry point may send a *false negative*. In other hand, when the last node of a specific type leaves its layer, the layer should just be removed. Likewise, the latency of the destruction message transmission to all entry points could lead to a *false positive*. However, these two cases remain seldom, and the period, during which these situations may arise, are quite short and only depend on the propagation speed of a message in the wireless network.

Below, we provide the anycast pseudo code algorithm:

Algorithm 1: ANYCAST(t)

Data: all information available (*cf.* Section 4.4)
Result: a couple $(id_t, (x_{id_t}, y_{id_t}))$
`[ Entry point coordinate computing ]`
1 $(x_e, y_e) \leftarrow$ **null**;
2 $d_e \leftarrow wsX \cdot wsY$;
3 $(x_h, y_h) \leftarrow (f_x(t), f_y(t))$;
4 **for** $i = 0$ **to** $m - 1$ **do**
5 **for** $j = 0$ **to** $n - 1$ **do**
6 $(x_{tmp}, y_{tmp}) \leftarrow ((i + x_h) \cdot csX, (i + y_h) \cdot csY)$;
7 **if** $d((x_{id}, y_{id}), (x_{tmp}, y_{tmp})) < d_e$ **then**
8 $(x_e, y_e) \leftarrow (x_{tmp}, y_{tmp})$;
9 $d_e \leftarrow d((id_x, id_y), (x_e, y_e))$;
`[ Contact the entry point and wait ]`
10 sendMessage((x_e, y_e), `Anycast`, t);
11 **return** *Reply from* (x_e, y_e);

[5]We call reliability the fact that a request of a functionality in *-cast suite is completed and the answer is up to date.

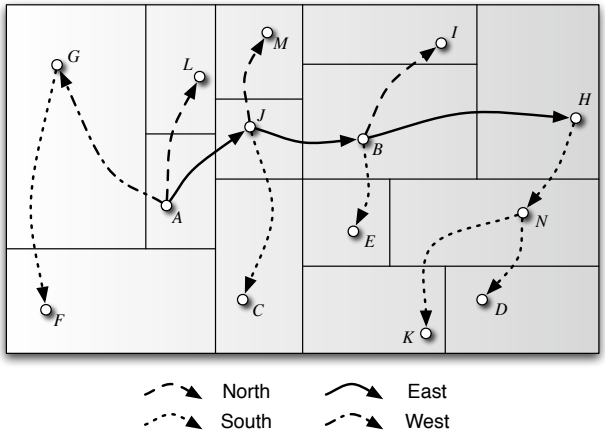

Figure 4: Example of broadcast in a LIGH-t-LAYER layer.

5.2 Broadcast functionality

The broadcast functionality ensures dissemination within a LIGH-t-LAYER. When a node initiates a broadcast operation, it first sends an anycast request to localize a type t contact node in the LIGH-t-LAYER, and forwards the broadcast request to this contact node.

In SOLIST, we leverage LIGH-t-LAYER structure in order to limit the redundancy of the broadcast. We ensure that no node will receive a message twice. As each node is responsible for a rectangle-shape area, in the following, we refer to the area frontiers by their direction (*North, South, East* and *West*) according to the virtual space and the node coordinates.

Figure 4 presents an example of the broadcast propagation in a LIGH-t-LAYER. We observe that this broadcast algorithm ensures that each node is reached once and only once, as opposed to the broadcast procedure in CAN which may impose some redundancy [16]. This intrinsically limits the resilience to failure. However, we consider that SOLIST, being aggressive in fixing the structure in case of failures, so message loss during a broadcast operation remain seldom. The energy saving gained by limiting unnecessary redundancy is definitely worth in a WSN. The algorithm follows the following steps:

(Step 1) The contact node (here, node A) forwards the message in the four directions to:

North and South *To each node aligned with the frontier.*
In Figure 4, A sends the message to L. However, A does not sends it to F as it is not aligned with its common frontier with A. Frontier alignment means that a node appear exactly in front of this frontier (for instance, consider the sub-area of F's, which correspond to the south extension of A area; F is not located in this sub-area);

East and West *To exactly one node in each direction, picked at random.*
In Figure 4, on this East side, A has the choice for sending the broadcast message between C and J. A chooses randomly to send the message to

J. On the West side, A does not have the choice and sends the message to G.

(Step 2) Each node receiving a message forwards it:

North *To each node aligned with the frontier if it receives the message from South, East or West;*
Represented by dash lines on Figure 4

South *To each node aligned with the frontier if it receives the message from North, East or West;*
Represented by point lines on Figure 4

East *To exactly one node picked at random if it receives the message from West;*
Represented by solid lines on Figure 4

West *To exactly one node picked at random if it receives the message from East.*
Represented by dash-point lines on Figure 4

Algorithm 2 presents the broadcast pseudo code algorithm, executed by nodes in the LIGH-t-LAYER. In this algorithm, N, S, E and W represent respectively the North, South, East and West directions. $isAlign(\cdot)$ returns a Boolean if the node is aligned with the common frontier, $loc(\cdot)$ returns the direction of the neighbour location and $getOne(\cdot)$ returns one instance picked at random in the provided set. Finally, $objectUpdate(\cdot)$ is provided by the application and depends of information to include in the broadcast reply, if needed.

Algorithm 2: BROADCAST(t)

Data: forwarder node id_f and Section 4.4 information
Result: if needed, an object o
`[ Forwarding nodes set etablishing ]`
$\Gamma \leftarrow \varnothing$;
foreach $n \in view_t[N]$ **do**
  if $isAling(n)$ **and** $loc(id_f) \neq N$ **then**
    $\Gamma \leftarrow \Gamma \cup \{n\}$;
foreach $n \in view_t[S]$ **do**
  if $isAling(n)$ **and** $loc(id_f) \neq S$ **then**
    $\Gamma \leftarrow \Gamma \cup \{n\}$;
if $loc(id_f) = E$ **then**
  $\Gamma \leftarrow \Gamma \cup \{getOne(view_t[W])\}$;
if $loc(id_f) = W$ **then**
  $\Gamma \leftarrow \Gamma \cup \{getOne(view_t[E])\}$;
`[ Forward the message ]`
sendMessage(Γ, `Broadcast`, t);
`[ Optional: update and send a reply ]`
foreach *Reply from* Γ **do**
  $objectUpdate(o)$;
sendMessage(id_f, `BroadcastReply`, o);

5.3 k-cast functionality

In order to provide a k-cast functionality in SOLIST, two possibilities could be considered, based on the non-redundant broadcast algorithm. The first one is a *probabilistic* k-cast, based on sending a j-cast search with $j < k$ for each direction of broadcast. Using such a probabilistic approach, it may occur some false negative, as the j value is determined probabilistically. Therefore, we preferred a *deterministic* approach for a better reliability and energy saving (indeed, in a deterministic way, at most k nodes are reached). However, this last approach imposes more delay as each node is reached sequentially.

The k-cast algorithm looks like a DFS (*Depth First Search*) in the LIGH-t-LAYER graph. The following heuristic according to the treatment order of directions is using: North → South → West → East

For instance, on Figure 4, node A receives a 10-cast request. It sends a 9-cast message first to L and waits its answer (only 9 because A is counting itself in the run). As L has no North neighbour, it sends directly its answer to A by decreasing to 8 the k value. As, A has no aligned South neighbour, it sends to one node across the West frontier and wait for its answer (here, node G). G does not have a North neighbour but have a South one. So, it sends the message to this one and waits the answer, *etc.* The 10-cast request from A follows the path below:

$$A \xrightarrow{k=9} L \xrightarrow{k=8} A \xrightarrow{k=8} G \xrightarrow{k=7} F \xrightarrow{k=6} G \xrightarrow{k=6} A$$
$$\xrightarrow{k=6} J \xrightarrow{k=5} M \xrightarrow{k=4} J \xrightarrow{k=4} C \xrightarrow{k=3} J \xrightarrow{k=3} B$$
$$\xrightarrow{k=2} I \xrightarrow{k=1} B \xrightarrow{k=1} E \xrightarrow{k=0} B \xrightarrow{k=0} J \xrightarrow{k=0} A$$

A can reply a `True` answer to the 10-cast initiator. The last ping-pong between B and its neighbours can be avoided, as B knows it have more than 3 neighbours.

Algorithm 3 presents the k-cast pseudo code algorithm, computed by nodes in the LIGH-t-LAYER. Functions used in this algorithm are defined as in Section 5.2.

We are now focusing on the evaluation of SOLIST architecture and service providing in the following section.

6 Evaluations

In this section, we present first the simulation environment, then we introduce the protocols used to compare SOLIST against, before presenting the SOLIST evaluation.

6.1 Simulation environment

In order to evaluate SOLIST, we used SeNSim, a software implemented for wireless sensor-based applications' simulation, developed by the ASAP/IRISA project team [1]. SeNSim is a Java software, which allows the creation of wireless sensor networks and analyses nodes behaviour under different events and failures scenarios. The simulator also allows the evaluation of the characteristics related to this architecture under different topologies, failures, and stimulus scenarios.

Algorithm 3: KCAST(t)

Data: forwarder node id_f, value k and Section 4.4 information
Result: if needed, an object o

`[ Forwarding nodes set etablishing ]`
$\Gamma \leftarrow \varnothing$;
foreach $n \in view_t[N]$ **do**
  if $isAling(n)$ **and** $loc(id_f) \neq N$ **then**
    $\Gamma \leftarrow \Gamma \cup \{n\}$;
foreach $n \in view_t[S]$ **do**
  if $isAling(n)$ **and** $loc(id_f) \neq S$ **then**
    $\Gamma \leftarrow \Gamma \cup \{n\}$;
if $loc(id_f) = E$ **then**
  $\Gamma \leftarrow \Gamma \cup \{getOne(view_t[W])\}$;
if $loc(id_f) = W$ **then**
  $\Gamma \leftarrow \Gamma \cup \{getOne(view_t[E])\}$;
`[ Forward the message to k nodes ]`
$\Gamma' \leftarrow \Gamma$;
while $k > 0$ **and** $\Gamma \neq \varnothing$ **do**
  $n \leftarrow getOne(\Gamma)$ *picked in order* N, S, W, E;
  sendMessage(n, `Kcast`, k);
  $\Gamma \leftarrow \Gamma - \{n\}$;
  $k \leftarrow$ *Reply from* n;
`[ Optional: update and send a reply ]`
foreach *Reply from* Γ' **do**
  $objectUpdate(o)$;
sendMessage(id_f, `KcastReply`, k, o);

In order to simulate large-scale sensor networks scenarios during a long period of time, the designed simulator is based on a discrete-event system.

We have simulated several topologies with the same system model, previously described in Section 2. We have used a light GPSR geographic routing

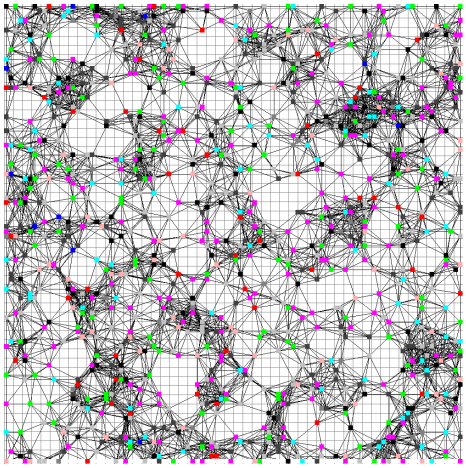

Figure 5: Example of network (1000 nodes, 10 types)

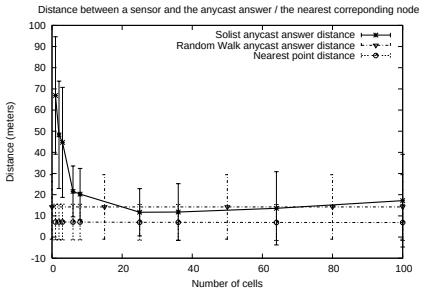

Figure 6: Average distance between the contact node known by an anycast request and the inquirer.

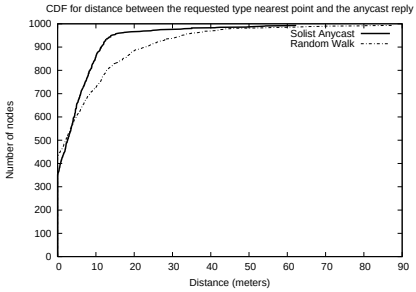

Figure 7: CDF for average distance between the contact node known by an anycast request and the inquirer for a 5×5 cells topology.

protocol [9], without the computation of *planar graph*. This last computation required non-negligible energy consumption. As we want to focus on SOLIST-only energy consumption, we only used the greedy routing with the *right-hand rules* in case of void density on a route, in the network[6].

Figure 5 presents an instance of a network topology, with 1,000 nodes spread into 10 static types (10 nodes of type 1, 30 of type 2, 50 of type 3, ..., 190 of type 10), in a 96×96 meters square zone, with a 0.7 meters transmission range. In the following, this topology, which is the worst case, is used for results that not present average values. For each workload presented in the following, each node joins the system, launches a broadcast and a k-cast, (so, by definition, several anycast) and finally leaves the network. Arrival and departure dates are randomly generated, as well as the request type for broadcast and k-cast. The k value is picked at random between 1 and 200, in order to get some `true` and `false` replies.

6.2 Presentation of the comparison algorithms

In order to evaluate SOLIST, we have to set our contribution against existing protocols. To achieve this comparison, we have used two well-known simple protocols, to compare two of the three members of the *-cast suite:

Anycast A simple but naive protocol to find one node of a specific type in an unstructured network is the *Random Walk* mechanism. The node sends a request to one of its direct neighbours (*i.e.*, in WSN context, one of the nodes in its transmission range). If this node belongs to the requested type, it sends to the initiator its position. Else, it chooses another random

[6] due to space constraint, we cannot develop these choice – see [9] for more information

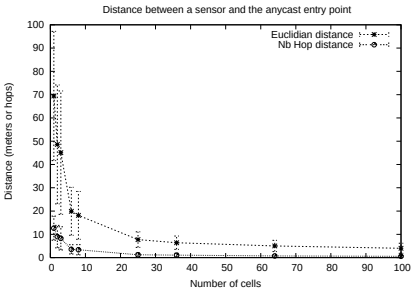

Figure 8: Average distance between the request node and the entry point.

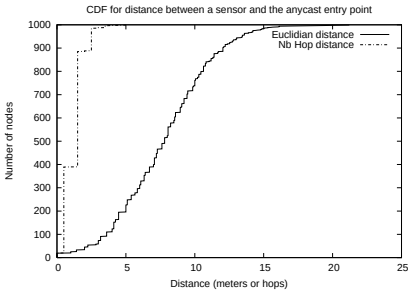

Figure 9: CDF for average distance between the request node and the entry point for 5×5 cells.

node among its neighbourhood, and sends the request to it, and so forth. If, after a predefined number of hops in the network, no node is found, the last request receiver sends a `NOT_FOUND` message to the initiator.

Broadcast The basic mechanism to reach each node of a type without any structure or information on the network is the *flooding-based* mechanism. Each node, receiving a broadcast message, forwards it to its entire neighbourhood. As multiple receptions can occur, a node will forward the message only one time. Thereby, a diffusion tree is thus dynamically constructed, and can be used for aggregate information in broadcast replies.

We don't propose k-cast comparison, because, to the best of our knowledge, no unstructured simple protocol has been proposed for this functionality.

6.3 SOLIST reliability

In this part, we evaluate the accuracy of SOLIST reply for each functionality. Energy consumption evaluation is presented in Section 6.4.

Anycast evaluation We evaluate the anycast mechanism in term of reply accuracy along two metrics. First, Figure 6 presents the average distance between the anycast query initiator and the contact node answered, for several numbers of cells configuration in SOLIST, and the respective standard deviation. The lower bound, corresponding to the average distance between the initiator and the nearest node of the request type, is represented by the *Nearest point distance* plot. Random walk anycast answer distance is also represented. For large size cells, corresponding to small number of cells in the system, anycast answers can be far away from the initiator, as the contact node replied is the nearest one from the entry point and not the initiator. Lower the cell size, nearer the

contact point. A side effect can be observed for huge number of cells, where the SOLIST anycast plot slightly grows up. Effectively, when a node joins a layer, it informs the nearest corresponding entry point. So, as the cells size decrease, some entry point may be nearer to the new node, without being aware of its arrival (we avoid broadcasting all entry points at each arrival to reduce energy consumption).

Figure 7 presents a cumulative distribution function (CDF) of this distance for a 5×5 cells configuration in SOLIST against an average random walk approach. In these simulations, more than 95 % of nodes with SOLIST have a distance less than 13.5 meters, while only 80 % fulfil this criterion when the random walk approach is used. Furthermore, the tail of SOLIST plot shows that all nodes have a distance less than 62.5 meters while the random walk mechanism results in a maximum distance of 89, which is about the size of the simulation environment.

Another interesting metric to evaluate the SOLIST anycast mechanism along, is the distance between a node and the nearest entry point. Figure 8 presents the average value of this distance and the standard deviation, according to the number of cells in SOLIST configuration. The distance is not only represented as in previous figures with the Euclidian distance, but also with the number of hops needed to reach the entry point. As predicted, the greater the number of cells, the smaller the cells size and the nearer the entry point. The interesting point is that from minimum 6 cells, the average number of hops is lower than 3 hops. This speaks to the low energy consumption for getting a contact node. Figure 9 presents a CDF of these two distances for a 5×5 configuration in SOLIST. 90 % of nodes can reach the nearest entry points in at most 1.5 hops[7], and 98.5 % by at most 2.5 hops. In this experiment, each cell has a 20×20 square meters size. No nodes are at a distance to an entry point of more than $\sqrt{2} \times 20$ meters (the diameter size of a cell) as predicted, but some are at a distance between $\sqrt{2} \times 20$ and $\frac{\sqrt{2} \times 20}{2}$: these nodes are the ones, which are located on the border of the network. They have fewer choices for entry point than nodes located in the centre of the network, having at least four entry points around them.

Broadcast and k-cast evaluation The reliability of such functionnalities is easy to analyse. We observe a Boolean behaviour: if the request leads to an answer, it means that it has reached all or k nodes respectively, else, the request won't lead to an answer. This may typically occur only during the LIGH-t-LAYER maintenance. In stable system behaviour, the hit ratio of broadcast or k-cast is 100 %. In case of departures or failures, the broadcast forwarding will be delayed until the maintenance step is over, in order to reach all nodes. The only case of unsuccessful request appears when a node has failed and its supervisor has not yet detected its failure. In this case, the request will be forwarded and the answer waited for endlessly. A timeout may be used to reemit the request in case of long time waiting, or ask for a reception acknowledgement for each message sent. The size of the network should then be accounted for to not forward another time the request in case of large run. This functionality has not been included in the simulation presented in this paper.

We are now focusing on the energy consumption evaluation.

[7] these results are given for a round trip message divided by 2 in order to take into account the non-symmetric route from a node to another with GPSR

Operation	nAh
Transmitting a packet	20.000
Receiving a packet	8.000
Radio listening for 1 millisecond	1.250
Flash Read Data	1.111
Flash Write/Erase Data	83.333

Table 1: Typical power requirement for various operations of a Mica mote proposed in [12].

6.4 Energy consumption

In order to evaluate and compare SOLIST against Flooding and Random Walk, we have run several simulations with the same network behaviour (as join date, leave date, failure date, number of events, *etc.*) Each simulation has a 10,000 discrete time length. Each node joins the network once and leaves or fails before the end of the simulation. Each node sends a broadcast and a k-cast request. As each join or *-cast request needs one anycast mechanism, these workloads consists in 3,000 anycast, 1,000 broadcast, 1,000 k-cast requests. Moreover, join, leave and maintenance consumption have to be taken into account. To imitate a real energy consumption, we use the power charcateristics described in Table 1, proposed in [12] from MICA nodes measurement. We consider that each node has a full battery of 2,200 mAh at the beginning of the simulation.

Figure 14 presents the energy consumption according to node identifier and time, for 2×4 cells configuration in SOLIST. This speaks for the load balancing in the network, as the great majority of node has consumed between 10 and 25 % for the whole simulation. Figure 10 presents for several cells configuration in SOLIST and random walks/flooding, the average total energy consumption and Figure 16 and 17 present for 2×4 cells configuration in SOLIST and random walk/flooding respectively the snapshot of the network according to the energy consumption in the topology presents in Figure 5, at the end of the simulation. The first one demonstrates the efficiency of SOLIST in saving energy for the whole network. The consumption in SOLIST is slightly increased according to the number of cells. This is due to the consumption needed for maintenance, as we see below. The two last figures show the load balancing characteristics in the network. We observe that the majority of node suffers from a lack of energy at the end of the simulation for random walk and flooding. On the contrary, SOLIST, using the same workload and an additional 1,000 k-cast requests, shows a maximum of energy consumption lower than 33 %. We observe high-density zones. Nodes in those zones are more solicited, due to the large number of message passing occurred during the simulation.

Figure 11 presents for several cells configurations in SOLIST and random walk, the average anycast energy consumption and Figure 18 and 19 present the end simulation energy snapshot respectively for SOLIST and Random Walk. The first one shows the interest of SOLIST by using at least 3 cells in this network topology. As tiny cells do not answer every time with the nearest contact point, the anycast mechanism increases slightly according to the cell size. Although, SOLIST anycast energy consumption required is no more than 35 % compare to the random walk mechanism. The two other figures present some compactness of

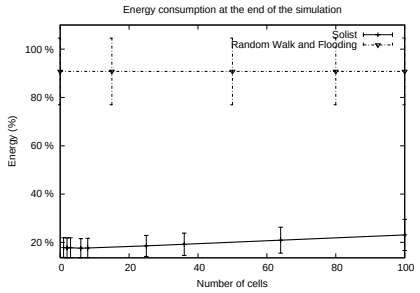

Figure 10: Average total energy consumption.

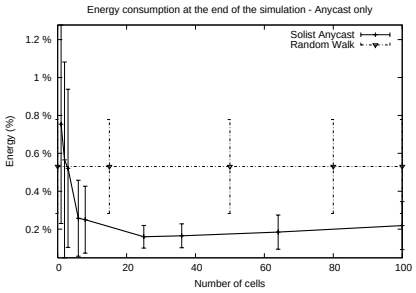

Figure 11: Average anycast energy consumption.

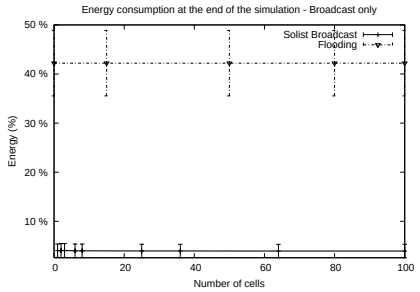

Figure 12: Average broadcast energy consumption.

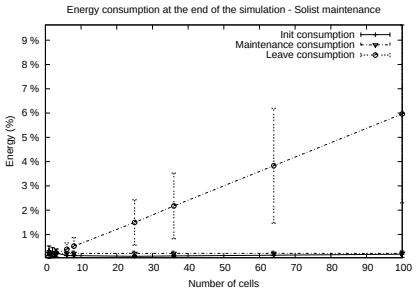

Figure 13: Average structure maintenance energy consumption.

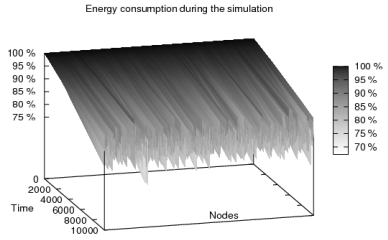

Figure 14: Node energy consumption according to time in the topology of Figure 5 with 2×4 cells.

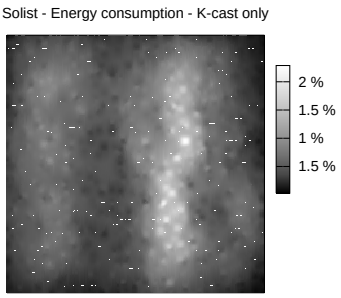

Figure 15: Energy consumption at the end of the simulation for k-cast in SOLIST in the topology of Figure 5 with 2×4 cells.

higher consumption points. For SOLIST anycast, these points correspond to the location of the 80 entry points in the system. We can easily infer the 8 cells from this figure. Contrarily, using the random walk mechanism, the point of higher consumption corresponds to the node with the highest density in the topology. This is due to the fact that requests have a higher probability to stay in this high-density neighbourhood (random walks have a low probability of visiting low-density neighbourhood). We also observe that the highest consomption in SOLIST is always lower than the ones observed suing random walks.

We now consider the consumption required by broadcasting requests . Figure 12 presents for several cells configurations in SOLIST and flooding, the average broadcast energy consumption. As the structure of LIGH-t-LAYER layers is not correlated with the number of cells in SOLIST, the average energy consumption for SOLIST broadcast and flooding is constant. As expected, this figure shows the interest of reaching only the required node instead of involving all nodes in the system. Figure 20 and 21 present the energy snapshot for broadcast in SOLIST and flooding in a 2×4 cells configuration. In SOLIST, only specific type nodes, corresponding to the most often contacted ones, show a higher energy consumption than the others (corresponding to white points in Figure 20). When flooding is applied, the whole network is involved in the process.

When broadcasting is applied to several types, it still remains more efficient to use several broadcast requests in SOLIST than one flooding operation. Despite that, if a total broadcast is requested (concerning *all* nodes in the network), the system sends a flooding dissemination instead of one broadcast for each type in the network.

Figure 15 presents, for k-cast mechanism in a 2×4 cells configuration in SOLIST, the snapshot of energy consumption at the end of the simulation. As opposed to broadcast consumption, k-cast consumption depends of the contact node returned by the anycast. In fact, as the k-cast mechanism needs to reach only a subset of nodes with a specific type and SOLIST anycast returns the nearest node around the entry point, it is expected to observe some compactness of higher consumption points around entry points, and larger than those appearing in Figure 18.

Finally, average structure maintenance energy requirement is presented in Figure 13 according to number of cells in SOLIST. As join operations and reorganizations in case of failure are only a local task, the associated energy consumption remains low, strictly lower than 0.25 % of the total amount of energy available for each one. Leaving operations require more energy when the number of cells is growing. This observation is a consequence of the linear expansion of leaving messages to inform each entry points. For instance, in 10×10 cells configuration, as each node leaves or fails during the simulation, 100,000 leaving messages are generated among the network...

The results show that SOLIST outperforms largely the two considered alternatives with respect to energy consumption. For the topology considered for this evaluation (*i.e.* 1,000 nodes among 10 types), an 8 cells configuration is ideal to obtain the best trade-off between efficiency and energy consumption.

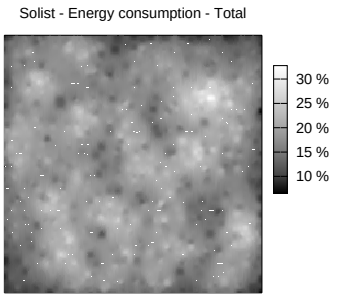

Figure 16: Energy consumption at the end of the simulation in SOLIST in the topology of Figure 5 with 2×4 cells.

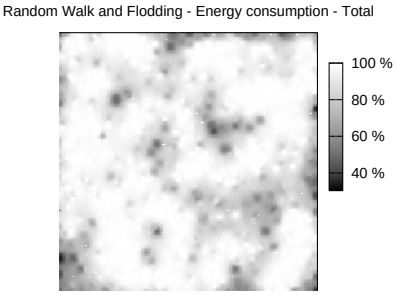

Figure 17: Energy consumption at the end of the simulation with random walk and flooding.

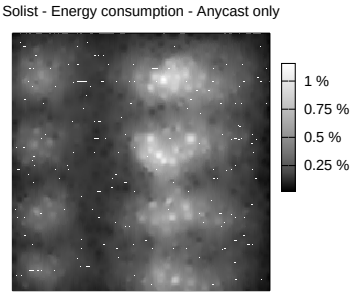

Figure 18: Energy consumption at the end of the simulation for anycast in SOLIST in the topology of Figure 5 with 2×4 cells.

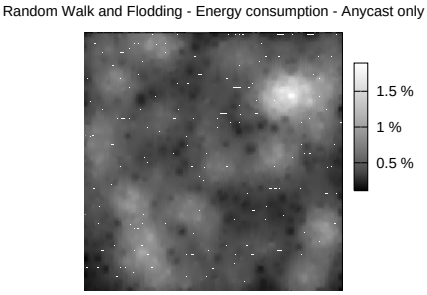

Figure 19: Energy consumption at the end of the simulation for anycast with random walk.

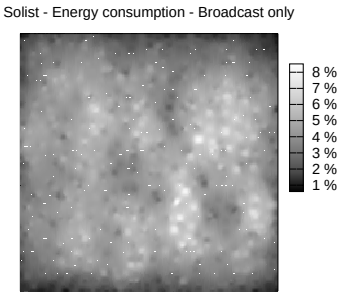

Figure 20: Energy consumption at the end of the simulation for broadcast in SOLIST in the topology of Figure 5 with 2×4 cells.

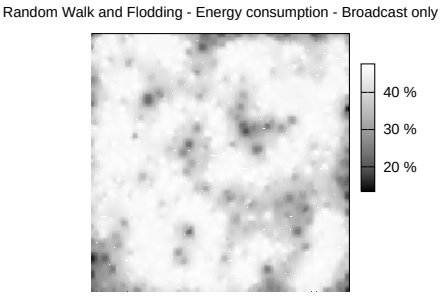

Figure 21: Energy consumption at the end of the simulation for broadcast with flooding.

7 Related works

In this section, we present various works related to our contribution. To the best of our knowledge, providing a common application-programming interface (API) for WSNs has not yet been proposed. Also, we are comparing the *-cast suite elements with previous works. These works address one of the several functionalities introduced in this paper.

On the first hand, we consider previous works on anycasting in WSNs. For instance, in [18], authors propose an anycast routing protocols based on hierarchical tree. These works used a base station in order to construct the routing tree. Although this is relevant in some contexts, one of SOLIST's advantages is the non-mandatory presence of base station in the system.

On the other hand, we can consider works on multicast in WSNs. This can obviously be compared to a broadcast for a specific type of sensors in SOLIST. Several works have been proposed recently: An interesting result is presented in [20]. This paper proposes a broadcast and a multicast mechanism for WSNs based on tree construction, for static networks. A tree-based structure has to be construct for each multicast group by suppresses useless links from the broadcast tree. This system has been evaluated on a small network though. Moreover, the system does not consider node failure or departures. Two important approaches in multicasting for WSNs are presented in [5, 8]. The first one [5] considers only reliability for any suitable multicast protocol, by lost message recovering and the second one [8] considers multicast on mobile sensors.

The nearest contribution has been proposed recently in [6]. Authors proposed a routing protocol for anycasting and multicasting in WSNs. They present a theoretical analysis of their scheme, based on dominance net construction. This paper presents interesting results and relies on a different model. Moreover, dynamic multicast group management requires the construction of a minimum spanning tree for every group modification. Finally, no simulation or experimentation has been done to illustrate the theoretical results. Then, comparison is hard as we do not provide such theoretical results.

Finally, we must cite a relevant approach to deal with information in a WSNs. TinyDB [11] is based on information acquisition directly on the network as us, but viewing the network as a physical database. Consequently, they propose a query language based on extension of SQL. As SOLIST is generic and represents an all-in-one solution for WSN application, TinyDB is application dependant according to query optimization and execution.

8 Conclusion

In this paper, we propose an effective all-in-one solution for the *-cast suite (*i.e.* anycasting, k-casting and multicasting) in static WSNs. This contribution, called SOLIST, is a generic lightweight system architecture for large-scale WSNs. SOLIST is composed of a finite set of overlays providing a common interface, with a type-based clustering. Based on this lightweight multi-layer structure (LIGHT-LAYER) and associated with a searching layer mechanism, SOLIST provides an efficient *-cast implementation in term of energy saving and reliability.

We evaluate by simulation each functionality provided by SOLIST and we compare SOLIST with two other standard algorithms: random walk for anycast

and tree-based flooding for broadcast. Results demonstrate that SOLIST outperforms these standard algorithms in term of energy saving and therefore provide a good trade-off between functionnality and energy consumption.

References

[1] SeNSim webpage `http://sensim.gforge.inria.fr/`, ASAP Research Project, INRIA Rennes, FR, 2005-2007.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.

[3] Y. Busnel, M. Bertier, E. Fleury, and A.-M. Kermarrec. GCP: Gossip-based Code Propagation for large-scale mobile wireless sensor networks. In *Proc. of Autonomics*, 2007.

[4] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *Proc. of Infocom*, 2003.

[5] R. Chandra, V. Ramasubramanian, and K. Birman. Anonymous Gossip: Improving multicast reliability in ad-hoc networks. In *Proc. of ICDCS*, 2001.

[6] R. Flury and R. Wattenhofer. Routing, anycast, and multicast for mesh and sensor networks. In *Proc. of Infocom*, 2007.

[7] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi. Meghdoot: content-based publish/subscribe over P2P networks. In *Proc. of Middleware*, 254–273, 2004.

[8] Q. Huang, C. Lu, and G.-C. Roman. Spatiotemporal multicast in sensor networks. In *Proc. of SenSys*, 205–217, 2003.

[9] B. Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *Proc. of MobiCom*, 2000.

[10] Y. Kwon, K. Mechitov, S. Sundresh, W. Kim, and G. Agha. Resilient localization for sensor networks in outdoor environments. In *Proc. of ICDCS*, 643–652, 2005.

[11] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database System*, 30(1):122–173, 2005.

[12] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proc. of WSNA 2002*, 88–97, 2002.

[13] R. O'Dell and R. Wattenhofer. Theoretical aspects of connectivity-based multi-hop positioning. *Theoretical Computer Science*, 344:47–68, 2005.

[14] S. Ratnasamy, D. Estrin, R. Govindan, B. Karp, S. Shenker, L. Yin, and F. Yu. Data-Centric Storage in Sensornets. In *Proc. of SIGCOMM*, 2002.

[15] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proc. of SIGCOMM*, 161–172, 2001.

[16] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proc. of NGC*, 2001.

[17] P. Rentala, R. Musunuri, S. Gandham, and U. Saxena. Survey on sensor networks. In *Proc. of MobiCom*, 2001.

[18] N. Thepvilojanapong, Y. Tobe, and K. Sezaki. HAR: Hierarchy-based Anycast Routing Protocol for Wireless Sensor Networks. In *Proc. of SAINT*, 204–212, 2005.

[19] S. Voulgaris, E. Rivière, A.-M. Kermarrec, and M. van Steen. Sub-2-Sub: Self-Organizing Content-Based Publish Subscribe for Dynamic Large Scale Collaborative Networks. In *Proc. of IPTPS*, 2006.

[20] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides. Energy-efficient broadcast and multicast trees in wireless networks. *Mobile Networks and Applications*, 7(6):481–492, 2002.

Contents

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399